



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/804,346	03/12/2001	Kenneth A. Chupa	CA920010012US1	1135
7590 12/29/2006				
A. Bruce Clay IBM Corporation T81/062 PO Box 12195 Research Triangle Park, NC 27709		EXAMINER YIGDALL, MICHAEL J		
		ART UNIT 2192	PAPER NUMBER	
SHORTENED STATUTORY PERIOD OF RESPONSE		MAIL DATE	DELIVERY MODE	
3 MONTHS		12/29/2006	PAPER	

**Please find below and/or attached an Office communication concerning this application or proceeding.**

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

**Office Action Summary**

Application No.

09/804,346

Applicant(s)

CHUPA ET AL.

Examiner

Michael J. Yigdal

Art Unit

2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

- 1) ☒ Responsive to communication(s) filed on 16 October 2006.
- 2a) ☒ This action is **FINAL**. 2b) ☐ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

- 4) ☒ Claim(s) 1-22 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-22 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

**Application Papers**

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on \_\_\_\_\_ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- \* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

- 1) ☐ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO/SB/08)  
Paper No(s)/Mail Date \_\_\_\_\_
- 4) ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_
- 5) ☐ Notice of Informal Patent Application
- 6) ☐ Other: \_\_\_\_\_

### **DETAILED ACTION**

1. This Office action is responsive to Applicant's submission filed on October 16, 2006.

Claims 1-22 are pending.

### ***Response to Arguments***

2. Applicant's arguments have been fully considered but they are not persuasive.

Applicant contends generally that the cited references fail to disclose that each physical code generator generates physical code corresponding to a specific focused performed operation (remarks, page 10, last paragraph).

However, as set forth in the Office action, Goldberg discloses that the physical code generator generates physical code corresponding to a specific query object (see, for example, column 12, lines 10-13). Goldberg further discloses that a specific query object performs specific operations (see, for example, column 5, lines 44-64). The operations are "focused" in the sense that every operation is focused on interacting with a database management system (see, for example, column 5, lines 44-64).

Applicant contends generally that the cited references fail to teach or suggest that each of the physical code generators in the generator dictionary generates physical code corresponding to a common target environment (remarks, page 11, first paragraph).

However, as set forth in the Office action, one example of a physical code generator in Goldberg is the "JDBCJavaQueryObjectImplGenerator" generator (see, for example, block 806 in FIG. 8). Goldberg discloses that the physical code generator generates physical code

Art Unit: 2192

corresponding to a common target environment, namely an environment comprising “a JAVA implementation with a JDBC database driver API set” (column 13, lines 16-18).

Applicant contends generally that none of the references teaches or suggests that a code generator is removed from the framework and replaced with a replacement code generator (remarks, page 11, second paragraph).

However, as set forth in the Office action, Goldberg in view of Pepin teaches or suggests that a code generator is replaced with a replacement code generator. For example, Goldberg discloses amending the generator dictionary to add a code generator (see, for example, column 13, lines 21-26), which suggests that one may amend the generator dictionary to replace a code generator as well. Pepin, in fact, discloses amending a generator dictionary to change or replace the implementation of a component, and illustrates an example in which the generator dictionary (metadata 330/430) is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3). The effect of the replacement is that designer 340/440 is removed from the generator dictionary (metadata 330/430). Thus, for the same reasons that replacing a code generator would have been obvious to one of ordinary skill in the art, so too would removing a code generator to replace the code generator have been obvious.

Finally, in response to Applicant’s statement that the Office action combines four different references (remarks, page 12, first paragraph), reliance on a large number of references in a rejection does not, without more, weigh against the obviousness of the claimed invention. See *In re Gorman*, 933 F.2d 982, 18 USPQ2d 1885 (Fed. Cir. 1991).

***Claim Rejections - 35 USC § 103***

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

4. Claims 1-5 and 11-22 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 6,496,833 to Goldberg et al. (art of record, "Goldberg") in view of U.S. Patent No. 6,948,150 to Pepin (art of record, "Pepin") in view of U.S. Patent No. 6,738,967 to Radigan (art of record, "Radigan") in view of U.S. Patent No. 6,742,175 to Brassard (art of record, "Brassard").

With respect to claim 1 (currently amended), Goldberg discloses a computer system for generating source code (see, for example, the abstract), said computer system comprising:

(a) a user amendable generator dictionary associating a generator routine with a generator identity, said generator identity identifying a code generator (see, for example, FIG. 8 and column 12, lines 32-50, which shows a programming construct or dictionary that associates a generator routine for a specific language and database with a generator class name or identity that identifies a code generator, and column 13, lines 21-26, which shows that the dictionary is amendable) and said generator dictionary comprising at least one logical generator and at least one physical code generator (see, for example, column 12, lines 18-27, which shows the

Art Unit: 2192

QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators).

Goldberg further discloses that each of the at least one physical code generators generates physical code corresponding to a specific focused performed operation (see, for example, column 12, lines 10-13, which shows that the code generator generates physical code corresponding to a specific query object, and column 5, lines 44-64, which further shows that the specific query object performs specific operations focused on interacting with a DBMS) in a common target environment (see, for example, column 13, lines 16-18, which shows the common target environment of a code generator comprising Java and JDBC database drivers).

Nonetheless, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator generates code corresponding to a specific component (see, for example, column 7, lines 51-65), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that each of the at least one physical code generators generates physical code corresponding to a specific performed operation, so that the implementation of the specific performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that each of the at least one physical code generators is independent from each other of the at least one physical code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that each of the at least one physical code generators is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

Although Goldberg does not expressly disclose the limitation wherein the generator dictionary is adapted to be amended by the user to remove at least one code generator and to replace the at least one code generator with a replacement code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

Goldberg further discloses:

(b) a code generation framework tool wherein said code generation framework tool, responsive to a request for an invocation of said generator routine, invokes said code generator identified by said generator identity associated with said generator routine (see, for example, column 11, lines 21-26, which shows a generator tool responding to input, i.e. to a request, and invoking the identified code generator);

wherein the at least one logical generator calls the at least one physical code generator to generate source code (see, for example, FIG. 8 and column 12, line 62 to column 13, line 20, which shows a logical generator such as `JavaQueryObjectImplGenerator` deriving or calling a physical code generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code).

With respect to claim 2 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a plurality of generator routines, each of said generator routines



Art Unit: 2192

associated with a generator identity (see, for example, FIG. 8 and column 12, lines 32-50, which shows a plurality of generator routines associated with a generator identity).

With respect to claim 3 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a text file (see, for example, column 12, lines 32-50, which shows that the dictionary is in the form of source code, inherently comprising a text file).

With respect to claim 4 (original), Goldberg further discloses the limitation wherein said generator routine comprises a logical generator name (see, for example, column 12, lines 32-50, which shows that the generator routine comprises a logical generator name such as “sybase\_ctlib” or “oracle\_oci”).

With respect to claim 5 (original), Goldberg further discloses the limitation wherein said code generation framework tool retrieves from said generator dictionary said generator identity responsive to said request (see column 11, lines 21-26, which shows that the generator tool selects the appropriate generator in response to the request).

With respect to claim 11 (currently amended), Goldberg discloses a method of generating source code for a first and a second deployment environment from a single input (see, for example, FIG. 8 and column 12, lines 9-51, which shows code generators for generating source code for a plurality of platforms or deployment environments), said method comprising:

(a) invoking a first code generator to generate source code for said first deployment environment from said single input, said first code generator identified by retrieving code generator identity data from a user amendable generator dictionary based on a generator routine

Art Unit: 2192

(see, for example, column 11, lines 21-26, which shows invoking the appropriate code generator based on input information, and column 12, lines 18-50, which shows retrieving the identity of a code generator from a generator dictionary based on a generator routine for the deployment environment, i.e. the first deployment environment, and column 13, lines 21-26, which shows that the dictionary is amendable).

Goldberg further discloses that the first code generator generates source code and corresponds to a specific focused performed operation of the single input in the first deployment environment (see, for example, column 12, lines 10-13, which shows that the code generator generates source code corresponding to a specific query object, and column 5, lines 44-64, which further shows that the specific query object performs specific operations focused on interacting with a DBMS).

Nonetheless, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator generates code corresponding to a specific component (see, for example, column 7, lines 51-65), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the first code generator generates source code corresponding to a specific performed operation of the single input in the first deployment environment, so that the implementation of the specific performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that the source code is generated independently of each other code generator.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the source code is generated independently of each other code generator, so that the code generators can be independently upgraded.

(b) modifying said generator dictionary to associate a second code generator with said generator routine (see, for example, column 13, lines 21-26, which shows adding new code generator classes, i.e. modifying the generator dictionary, to associate other code generators).

Although Goldberg does not expressly disclose removing the first code generator and replacing the first code generator the second code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18).

Art Unit: 2192

Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

(c) invoking said second code generator to generate source code for said second deployment environment from said single input, said second code generator identified by retrieving code generator identity data from said generator dictionary based on said generator routine (see, for example, column 11, lines 21-26, which shows invoking the appropriate code generator based on input information, and column 12, lines 18-50, which shows retrieving the identity of a code generator from the generator dictionary based on a generator routine for the deployment environment, i.e. the second deployment environment), wherein the second code generator generates source code corresponding to the specific focused performed operation of the single input in the second deployment environment (see the explanation above).

With respect to claim 12 (original), Goldberg further discloses the limitation wherein said invoking said first code generator comprises a call issued by one of a code generation framework tool and a code generator; and wherein said invoking said first code generator comprises a call issued by one of said code generation framework tool and a code generator (see, for example, column 11, lines 21-26, which shows invoking or calling the appropriate code generator).

With respect to claim 13 (original), Goldberg further discloses the limitation wherein said modifying comprises editing said generator dictionary (note that modifying the generator dictionary inherently comprises editing the generator dictionary).

With respect to claim 14 (currently amended), Goldberg discloses a generator dictionary stored on a recordable medium comprising:

a plurality of generator routines that include at least one logical generator and at least one physical code generator (see, for example, column 12, lines 18-27, which shows the QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators), each of said generator routines associated with code generator identity data (see, for example, FIG. 8 and column 12, lines 32-50, which shows a programming construct or dictionary comprising a plurality of generator routines for specific languages and databases associated with code generator class names or identities).

Goldberg further discloses that each of the plurality of generator routines generates source code corresponding to a specific focused performed operation (see, for example, column 12, lines 10-13, which shows that the code generator generates source code corresponding to a specific query object, and column 5, lines 44-64, which further shows that the specific query

Art Unit: 2192

object performs specific operations focused on interacting with a DBMS) in a common target environment (see, for example, column 13, lines 16-18, which shows the common target environment of a code generator comprising Java and JDBC database drivers).

Nonetheless, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator generates code corresponding to a specific component (see, for example, column 7, lines 51-65), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that each of the plurality of generator routines generates source code corresponding to a specific performed operation, so that the implementation of the specific performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that the at least one physical code generator is independent from each other of the at least one physical code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the at least one physical code generator is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

wherein the at least one logical generator calls the at least one physical code generator to generate source code (see, for example, FIG. 8 and column 12, line 62 to column 13, line 20, which shows a logical generator such as `JavaQueryObjectImplGenerator` deriving or calling a physical code generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code), and

wherein the generator dictionary is designed to be amended by a user (see, for example, column 13, lines 21-26, which shows that the dictionary is amendable).

Although Goldberg does not expressly disclose removing at least one of the plurality of generator routines and replacing the at least one of the plurality of generator routines with a replacement generator routine, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

With respect to claim 15 (currently amended), Goldberg discloses a code generation framework tool (see, for example, the abstract) comprising:

(a) a receiver for receiving input data (see, for example, GUI 634 in FIG. 6 and column 10, lines 47-53, which shows an interface or receiver for receiving input data);

(b) a user amendable generator dictionary accessor for retrieving data from a generator dictionary comprising at least one logical generator and at least one physical code generator (see, for example, column 12, lines 29-51, which shows an accessor method for retrieving data from a generator dictionary, and column 12, lines 18-27, which shows the QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators, and column 13, lines 21-26, which shows that the dictionary is amendable).

Goldberg does not expressly disclose that the at least one physical code generator is independent from each other of the at least one physical code generators.



However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the at least one physical code generator is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

(c) an invoking mechanism for calling a code generator (see, for example, code generator 604 in FIG. 6 and column 11, lines 21-26, which shows invoking a code generator); and

wherein, responsive to a receipt of input data at said receiving, said invoking mechanism calls a code generator identified by identity data retrieved by said generator dictionary accessor from a generator dictionary (see, for example, column 11, lines 21-26, which shows invoking a code generator in response to input data, and column 12, lines 18-51, which shows that the identity of a code generator is retrieved from the accessor method).

Goldberg further discloses that the identified code generator corresponds to a specific focused performed operation of the input data in a target environment (see, for example, column 12, lines 10-13, which shows that the code generator corresponds to a specific query object, and column 5, lines 44-64, which further shows that the specific query object performs specific operations focused on interacting with a DBMS).

Nonetheless, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator corresponds to a specific component (see, for example, column 7, lines 51-65), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the identified code generator corresponds to a specific performed operation of the input data, so that the implementation of the specific performed operation can be changed independently of and without modifying the generator tool itself.

Although Goldberg does not expressly disclose the limitation wherein the generator dictionary is adapted to be amended by the user to remove the code generator and replace the code generator with a replacement code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

With respect to claim 16 (original), Goldberg further discloses a data dictionary associating a generator routine with identity data identifying a code generator (see, for example, column 12, lines 9-51, which shows a programming construct or dictionary that associates a generator routine for a specific language and database with a generator class name or identity, which identifies a code generator).

With respect to claim 17 (previously presented), Goldberg further discloses the limitation wherein said generator dictionary accessor identifies a generator routine within said input data received and wherein said code generator identified is determined by retrieving said identity data associated with said generator routine identified (see, for example, column 12, lines 29-51, which shows the accessor method for identifying a code generator based on a generator routine, which is determined from the input data specifying a language and database).

With respect to claim 18 (currently amended), Goldberg discloses a computer readable medium storing instructions and data (see, for example, column 20, lines 5-11), said instructions and data for adapting a computer system to:

(a) responsive to a request for invoking a generator routine, identify, in a user amendable generator dictionary that includes at least one logical generator and at least one physical code generator, a code generator associated with said generator routine (see, for example, column 11, lines 21-26, which shows selecting the appropriate generator in response to a user request, and column 12, lines 18-50, which shows identifying the code generator from a generator dictionary, including the QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators, and column 13, lines 21-26, which shows that the dictionary is amendable).

Goldberg further discloses that the identified code generator corresponds to a specific focused performed operation of the input data (see, for example, column 12, lines 10-13, which shows that the code generator corresponds to a specific query object, and column 5, lines 44-64, which further shows that the specific query object performs specific operations focused on interacting with a DBMS).

Nonetheless, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator corresponds to a specific component (see, for example, column 7, lines 51-65), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the identified code generator corresponds to a specific performed operation of the input data, so that the implementation of the specific performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that the at least one physical code generator is independent from each other of the at least one physical code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32)..

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the at least one physical code generator is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

Art Unit: 2192

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

(b) pass said input data to said code generator identified (see, for example, column 11, lines 7-20, which shows input data specifying the operating environment, and column 12, lines 18-27, which shows passing the data to a generator method), said code generator being operable to:

call another code generator to generate the source code (see, for example, column 12, line 62 to column 13, line 20, which shows a generator such as `JavaQueryObjectImplGenerator` deriving or calling another generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code); and

generate the source code (see, for example, column 11, lines 21-26, which shows generating source code).

With respect to claim 19 (previously presented), the limitations recited in the claim are analogous to those of claim 7 (see the rejection of claim 7 below).

With respect to claim 20 (previously presented), the limitations recited in the claim are analogous to those of claim 8 (see the rejection of claim 8 below).

With respect to claim 21 (previously presented), the limitations recited in the claim are analogous to those of claim 9 (see the rejection of claim 9 below).

With respect to claim 22 (previously presented), the limitations recited in the claim are analogous to those of claim 10 (see the rejection of claim 10 below).

5. Claims 6 is rejected under 35 U.S.C. 103(a) as being unpatentable over Goldberg in view of Pepin in view of Radigan.

With respect to claim 6 (currently amended), Goldberg discloses a method for generating source code from input data (see, for example, the abstract), said method comprising:

(a) responsive to a request for invoking a generator routine, identifying from a plurality of code generators a code generator associated with said generator routine (see, for example, column 11, lines 21-26, which shows selecting the appropriate generator in response to a user request).

Goldberg further discloses that the identified code generator corresponds to a specific focused performed operation of the input data (see, for example, column 12, lines 10-13, which shows that the code generator corresponds to a specific query object, and column 5, lines 44-64, which further shows that the specific query object performs specific operations focused on interacting with a DBMS) in a common target environment (see, for example, column 13, lines 16-18, which shows the common target environment of a code generator comprising Java and JDBC database drivers).

Nonetheless, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator corresponds to a specific component (see, for example, column 7, lines 51-65), so that the implementation of the

Art Unit: 2192

component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the identified code generator corresponds to a specific performed operation of the input data, so that the implementation of the specific performed operation can be changed independently of and without modifying the generator tool itself.

(b) passing said input data to said code generator identified (see, for example, column 11, lines 7-20, which shows input data specifying the operating environment, and column 12, lines 18-27, which shows passing the data to a generator method), said code generator being operable to:

call another code generator to generate the source code (see, for example, column 12, line 62 to column 13, line 20, which shows a generator such as `JavaQueryObjectImplGenerator` deriving or calling another generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code); and

generate the source code (see, for example, column 11, lines 21-26, which shows generating source code).

Goldberg does not expressly disclose that the source code is generated independently of other code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer



Art Unit: 2192

(see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the source code is generated independently of other code generators, so that the code generators can be independently upgraded.

6. Claims 7-10 are rejected under 35 U.S.C. 103(a) as being unpatentable over Goldberg in view of Pepin in view of Radigan, as applied to claim 6 above, and further in view of Brassard.

With respect to claim 7 (previously presented), Goldberg further discloses the limitation wherein said identifying comprises retrieving from a user amendable generator dictionary code generator identity data associated with said generator routine (see, for example, FIG. 8 and column 12, lines 27-51, which shows a programming construct or dictionary for retrieving the identity of a code generator associated with a generator routine that represents a specific language and database, and column 13, lines 21-26, which shows that the dictionary is amendable).

Although Goldberg does not expressly disclose the limitation wherein the generator dictionary is adapted to be amended by the user to remove at least one of the plurality of code generators and to replace the at least one of the plurality of code generators with a replacement code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of

Art Unit: 2192

Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

With respect to claim 8 (original), Goldberg further discloses the limitation wherein said identifying further comprises prior to said retrieving, locating said generator routine in said generator dictionary (see, for example, column 12, lines 32-50, which shows a "switch" construct, which inherently involves locating the appropriate "case" statement before returning the identity of a code generator).

With respect to claim 9 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a lookup table (see, for example, column 12, lines 32-50, which shows a programming construct or dictionary that serves as a lookup table).

With respect to claim 10 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a text file (see, for example, column 12, lines 32-50, which shows that the dictionary is in the form of source code, inherently comprising a text file).

### *Conclusion*

7. Applicant's amendment necessitated any new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

8. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Michael J. Yigdall whose telephone number is (571) 272-3707. The examiner can normally be reached on Monday through Friday from 7:30am to 4:00pm.

Art Unit: 2192

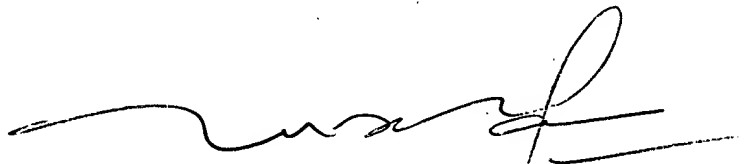
If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

MY

Michael J. Yigdall  
Examiner  
Art Unit 2192

mjy



**TUAN DAM**  
**SUPERVISORY PATENT EXAMINER**